19CST303 – NNDL

Gradient descent

1. Introduction

Gradient descent (GD) is an iterative first-order optimisation algorithm, used to find a local minimum/maximum of a given function. This method is commonly used in machine learning (ML) and deep learning (DL) to minimise a cost/loss function (e.g. in a linear regression). Due to its importance and ease of implementation, this algorithm is usually taught at the beginning of almost all machine learning courses.

However, its use is not limited to ML/DL only, it's widely used also in areas like:

- control engineering (robotics, chemical, etc.)
- computer games
- mechanical engineering

That's why today, we will do a deep dive into the math, implementation and behaviour of first-order gradient descent algorithm. We will navigate the custom (cost) function directly to find its minimum. That means there will be no underlying data like in typical ML tutorials — we will be more flexible regarding a function's shape.

This method was proposed long before the era of modern computers by Augustin-Louis Cauchy in 1847. Since that time, there was an significant development in computer science and numerical methods. That led to numerous improved versions of Gradient Descent. However, in this article we're going to use a basic/vanilla version implemented in Python.

2. Function requirements

Gradient descent algorithm does not work for all functions. There are two specific requirements. A function has to be:

- **differentiable**
- **convex**

First, what does it mean it has to be **differentiable**? If a function is differentiable it has a derivative for each point in its domain — not all functions meet these criteria. First, let's see some examples of functions meeting this criterion:
of differentiable functions; Image by author

Typical non-differentiable functions have a step a cusp or a discontinuity:

Next requirement — **function has to be convex**. For a univariate function, this means that the line segment connecting two function's points lays on or above its curve (it does not cross it). If it does it means that it has a local minimum which is not a global one.

The value of this expression is zero for x=0 and x=1. These locations are called an inflexion point — a place where the curvature changes sign — meaning it changes from convex to concave or vice-versa. By analysing this equation we conclude that :

- for x<0: function is convex
- for 0<x<1: function is concave (the 2nd derivative < 0)
- for x>1: function is convex again

Now we see that point x=0 has both first and second derivative equal to zero meaning this is a saddle point and point x=1.5 is a global minimum.

Let's look at the graph of this function. As calculated before a saddle point is at x=0 and minimum at x=1.5.

3. Gradient

Before jumping into code one more thing has to be explained — what is a gradient. Intuitively it is a slope of a curve at a given point in a specified direction.

In the case of **a univariate function**, it is simply the **first derivative at a selected point**. In the case of **a multivariate function**, it is a **vector of derivatives** in each main direction (along variable axes). Because we are interested only in a slope along one axis and we don't care about others these derivatives are called **partial derivatives**.

A gradient for an n-dimensional function f(x) at a given point p is defined as follows:

The upside-down triangle is a so-called nabla symbol and you read it “del”. To better understand how to calculate it let’s do a hand calculation for an exemplary 2-dimensional function below.

4. Gradient Descent Algorithm

Gradient Descent Algorithm iteratively calculates the next point using gradient at the current position, scales it (by a learning rate) and subtracts obtained value from the current position (makes a step). It subtracts the value because we want to minimise the function (to maximise it would be adding). This process can be written as:

There’s an important parameter **η** which scales the gradient and thus controls the step size. In machine learning, it is called **learning rate** and have a strong influence on performance.

- The smaller learning rate the longer GD converges, or may reach maximum iteration before reaching the optimum point
- If learning rate is too big the algorithm may not converge to the optimal point (jump around) or even to diverge completely.

In summary, Gradient Descent method’s steps are:

1. choose a starting point (initialisation)
2. calculate gradient at this point

3. make a scaled step in the opposite direction to the gradient (objective: minimise)

4. repeat points 2 and 3 until one of the criteria is met:

- maximum number of iterations reached
- step size is smaller than the tolerance (due to scaling or a small gradient).

    - This function takes 5 parameters:
    - 1. **starting point** [float] - in our case, we define it manually but in practice, it is often a random initialisation
    - 2. **gradient function** [object] - function calculating gradient which has to be specified before-hand and passed to the GD function
    - 3. **learning rate** [float] - scaling factor for step sizes
    - 4. **maximum number of iterations** [int]
    - 5. **tolerance** [float] to conditionally stop the algorithm (in this case a default value is 0.01)